Achieving Efficient Access Control via XACML Policy in Cloud Computing

Xin Pei, Huiqun Yu, Guisheng Fan
Department of Computer Science and Engineering
East China University of Science and Technology
Shanghai 200237, China
Email: yhq@ecust.edu.cn

***Abstract*—One primary challenge of applying access control methods in cloud computing is to ensure data security while supporting access efficiency, particularly when adopting multiple access control policies. Many existing works attempt to propose suitable frameworks and schemes to solve the problems, however, these proposals only satisfy specified use cases. In this paper, we take XACML as the policy language and build up a logical model. Based on this, we introduce the fine-grained data fragment algorithm to optimize the policies, whose *resource* property represents physical meaningful data blocks. Data are organized in a tree structure, where each leaf node represents a minimal physical meaningful data block, and internal nodes are combined data types. This method can eliminate conflicts and redundancies among rules and policies, thus to refine the policy set and achieve fine-grained access control. Our approach can also be applied to processing multi-types of data, and experiments are carried out to show the improvements of efficiencies.**

Keywords-Access control; Policy optimization; Data fragment; XACML; cloud computing

I. INTRODUCTION

In the last several years, cloud computing brings us great convenience on data outsourcing by providing nearly unlimited storage resources on demand [1]. This allows content providers to create, manage, and control the personal data remotely with high efficiency. Moreover, the charge manner of cloud service is pay-as-you-use which costs relatively lower prices compared with self-maintenance. As promising as it is, cloud storage service also involves many challenges [2], such as the problems of fine-grained access control on multiple data types and the confidentiality of private data. The traditional access control methods are only applicable to rigid data objects, and the policy decision on a request results in either *permit* or *deny*.

Motivated by the requirements of high performance and flexible access control, XACML (eXtensible Access Control Mark-up Language) [3] is proposed to solve data access problem in cloud computing. XACML is an XML-based language, and it contains a hierarchical logic model which is applied to a particular decision request in access control policies for Web applications and Web services. Meanwhile, XACML offers a large set of built-in functions, data types, combining algorithms, and standard profiles for defining application-specific features. There are lots of prior works on applying XACML as access control policy, which focus on policy attesting, conflict detection and policy optimization, etc. Mont and Pearson propose the 'sticky policy' based on XACML to facilitate access control for outsourced data [4] [5], and Trabelsi extends this policy to the cloud environment [6]. However, their proposals only focus on the system framework and the shared data is considered in a single type. Hu and Ahn introduce a description logic (DL)-based policy management approach for Web access control policies, they adopt Answer Set Programming (ASP) to formulate XACML [7], and they further propose a method for conflict detection and resolution in [8]. However, DL cannot fully cover XACML semantics, and it fails to handle complex comparisons, multi-type of decisions as well as combining algorithms. Wang and Feng propose a rule redundant elimination method based on related types of hierarchical attributes tree and provide an XACML policy optimization engine [9], but the access efficiency depends on the amount of rules. Said and Shehab propose a framework for policy evaluation [10], and Lin and Rao suggest a similarity measurement technique among policies [11]. Meanwhile, Bertolino and Daoudagh propose an automated testing method for XACML [12]. Their works are worth well in policy attesting and measurement, but they do not achieve a practical scheme for optimizing policy with consideration of fine-grained access control. Many practical models of XACML are built in [13] [14] [15], but these models are not considerate in decision efficiency, especially for large scale of policies.

Compared with existing policy models, XACML is more comprehensive and intuitive for applying to cloud access control. In this paper, we analyze the logic model of XACML by taking account of all its components and internal functions. Based on this model, we propose the data fragmentation and policy refinement algorithms via building up a three-layers resource access tree, so as to achieve fine-grained access control over multi-types of outsourced data. In the end, we discuss a case study on healthcare records management, and the performances are illustrated by experiments using XACML tools.

II. XACML ANALYSIS

XACML is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) in 2003. In XACML, the complete policy applicable to a particular decision might be composed of a number of individual rules, policies and policy sets, in which there exists a target expression as the criteria for incoming requests, and

(DOI Reference Number: 10.18293/SEKE2015-037)

all these elements are organized in a hierarchical order [16]. To render an authorization, it must be possible to combine multi-rules to form the single decision applied to the request, and the final decision is either made by the rule 'effect' or the combined decisions of children rules and policies.

A. XACML elements

We define the main elements *policy set*, *policy*, *rule*, *target*, *request*, *effect* and *combining algorithm* (*CA* for short) of XACML syntax as follows, where the values $'PO', 'DO', 'FA', 'OOA'$ represent for 'permit-override', 'deny-override', 'first-applicable' and 'only-one-applicable', respectively.

$$Policyset ::= < target >, \{< Policyset > | < Policy >\}^{+}, < CA >$$

$$Policy ::= < target >, \{< rule >\}^{+}, < CA >$$

$$rule ::= < target >, < effect >, < condition >, < obligation >$$

$$target ::= \{< attr\_type >, < attr\_value >, < match\_id >\}^{*}$$

$$request ::= \{< attr\_type >, < attr\_value >\}^{+}$$

$$effect ::= 'permit' | 'deny'$$

$$CA ::= 'PO' | 'DO' | 'FA' | 'OOA'$$

Additionally, XACML extends the decision values by appending two extra status 'not-applicable' and 'indeterminate', based on the previous policy languages with only 'permit' and 'deny'. The status 'not-applicable' represents that the request does not match any rule in the designated policy, while 'indeterminate' indicates errors in the matching procedure (e.g., The request does not match a 'critical' attribute in the target).

B. Decision principles

We describe the principles of XACML to make a decision. On receiving a request, the policy decision point (PDP) executes *target matching* and results *MR* in the set $V_{MR} = \{'T', 'F', 'IN'\}$, representing 'match', 'un-match' and 'indeterminate' respectively. If this matching procedure happens in a rule, it leads to a decision according to the rule 'effect' and *MR*. Otherwise, if the *target matching* belongs to a policy or policy set, the final decision is integrated by the combining algorithm affecting on children rules and policies.

We denote a user request as a vector $req = \{a_1, a_2, ..., a_n\}$, each a_i in *req* is an attribute value which belongs to the attribute type A_i pre-defined in XACML. The principles are listed as below.

1) *Target matching*. The connection of elements in a target can be either '*AllOf*' or '*AnyOf*', which indicates the operations of AND | OR.

Assume a request matches with *K* elements in the target, formally, $req(K): A_1 \times A_2 \times ... \times A_K \to V_{MR}$. The '*AllOf*' property performs as in (1), and the '*AnyOf*' property performs as in (2).

$$\bigcap_{i=1}^{K} m_i = m_1 \wedge m_2 \wedge ... \wedge m_K = \begin{cases} T, & if\ \forall i \in [1, K], m_i = T \\ F, & if\ \exists i \in [1, K], m_i = F \\ IN, & error \end{cases} \quad (1)$$

$$\bigcup_{i=1}^{K} m_i = m_1 \vee m_2 \vee ... \vee m_K = \begin{cases} T, & if\ \exists i \in [1, K], m_i = T \\ F, & if\ \forall i \in [1, K], m_i = F \\ IN, & error \end{cases} \quad (2)$$

2) *Rule decision*. Regardless of obligation property, a rule can be abbreviated as $rule(t, e, c)$, where *t*, *e*, *c* are 'target', 'effect' and 'condition'. The effect domain of a rule is $E = \{'permit', 'deny'\}$. The condition is a list of constraints that request must satisfy, and it shares the same matching result domain V_{MR} with target. We denote the rule decision domain as $V_D = \{'P', 'D', 'N', 'IN'\}$, corresponding to 'Permit', 'Deny', 'Not-applicable' and 'Indeterminate' respectively. Thus, the mapping of rule decision domain can be represented as $rule(t, e, c): V_{MR} \times E \times V_{MR} \to V_D$, and the decision is illustrated in (3).

$$rule(t, e, c) = \begin{cases} P & if\ t \wedge c = T\ and\ e = 'permit' \\ D & if\ t \wedge c = T\ and\ e = 'deny' \\ N & if\ t \wedge c = F \\ IN & error \end{cases} \quad (3)$$

3) *Policy/Policy set decision*. Denoting a policy as $policy\left(t, CA(\{rule\}^{+})\right)$ and a policy set as $policyS\left(t, CA(\{policy \mid policyS\}^{+})\right)$. The combing algorithms is a set $CA = \{'PO', 'DO', 'FA', 'OOA'\}$ that operates on decision domain V_D, it combines all the decisions made by children rules and policies into one final decision. Formally, let *S* be a set, $CA(S): \{V_D\}^{|S|} \to V_D$. Therefore, the policy and policy set are similar and can be formalized as $policy\left(t, CA(S)\right): V_{MR} \times V_D \to V_D$. According to different combing algorithms, the policy and policy set decision are concluded in (4).

$$policy\left(t, CA(S)\right) = \begin{cases} CA(S) & if\ t = T \\ N & if\ t = F\ or\ t = IN\ and\ CA(S) = N \\ IN & if\ t = IN \end{cases} \quad (4)$$

C. A sample XACML

Fig. 1 illustrates a simple XACML policy example P_1, containing three rules r_1, r_2, r_3. In this figure, we use brief XML syntax to describe the policy rather than standard XACML format. The resource property in rules reflects to physical data, and the algorithm is mainly constructed on the resource dimension. In this policy, four resources RS1, RS2, RS3 and RS4 are considered, and they have intersections on which rules may conflict and have redundancies.

```
<policy policyID="P1" CA="Deny-Overrides">
  <target>
        <Actions>Read, Write</Actions>
  </target>
  <Rule RuleID="r1" Effect="Permit">
    <target>
        <Subjects>Alice, Bob</Subjects>
        <Resources>RS1, RS2</Resources>
        <Actions>Read, Write</Actions>
    </target>
    <Condition> 8:00<= Time <=12:00 </Condition>
  </Rule>
  <Rule RuleID=" r2" Effect="Permit">
    <target>
      <Subjects>Bob</Subjects>
        <Resources>RS4</Resources>
        <Actions>Read</Actions>
    </target>
  </Rule>
  <Rule RuleID=" r3" Effect="Deny">
    <target>
        <Subjects> Bob, Jim</Subjects>
        <Resources>RS2, RS3</Resources>
        <Actions>Write</Actions>
    </target>
    <Condition>9:00<= Time <=15:00 </Condition>
  </Rule>
</policy>
```

Figure 1. A simple XACML example.

We can formalize the rules into Boolean expressions, for example, r_1 is illustrated in (5).

$$\begin{aligned} BoolExpression_{r_1} = & (Subject = 'Alice' \vee 'Bob') \\ & \wedge (Resource = 'RS1' \vee 'RS2') \\ & \wedge (Action = 'Read' \vee 'Change') \\ & \wedge (AnyCondition) \end{aligned} \tag{5}$$

Assuming a user Bob makes a request for writing to RS2 at 10:00 am. Formally, $req(Subject = 'Bob', Resource = 'RS2', Action = 'Write', Condition = '10:00\ am.')$. On execution, the target of P_1 matches the request and returns *T*. Then, r_1 checks its target and conditions, and gives out the 'permit' decision as defined in *effect*. However, it continues to match the next rules because the CA of parent policy P_1 is 'Deny-override'. Accordingly, r_2 does not match the request (outputs 'N' as not-applicable) while r_3 returns 'deny' as its rule decision. Finally, the policy combines the three results and gives the 'deny' decision according to its CA.

III. Fine-grained Policy Optimization Algorithm

In this section, we introduce a data fragment algorithm for resource isolation and policy refinement. We build up a three-layer structure of resources, and map the effective policies to each leaf data node so that fine-grained access control is achieved.

A. Data fragmentation

Firstly, we give the definition of *Disjoint set*, based on which we execute the policy projection algorithm.

Definition 1(*Disjoint set*) Let $S\{s_1, s_2, ..., s_n\}$ be a resource set. If $\{\neg\exists res : res \in s_i, res \in s_j\}$ and any operation on s_i will not affect $s_j (i \neq j)$, S is a disjoint set.

Taking the XACML policy in Fig. 1 as example, the four resources (RS1, RS2, RS3, RS4) intersect with each other as shown in Fig. 2. In order to obtain a disjoint resource set $RS(s1, s2, s3, s4, s5, s6)$, we introduce the data fragment algorithm in Algorithm 1.

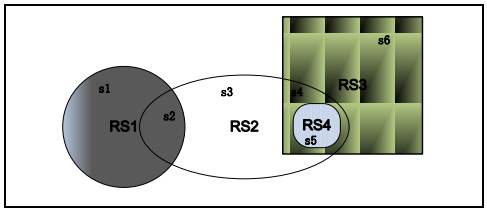


Figure 2. Relationship of resources.

Algorithm 1: Data fragment algorithm

```
// INPUT: a policy.
// OUTPUT: a disjoint set.
Project(policy)
    for each res ∈ GetResources(policy) do
        for each s ∈ RS do
            if res ⊂ s then
                RS.add(s\res);
                RS.replace(s, res); break;
            else if res ⊃ s then
                RS.replace(res, res\s); break;
                 else if res ∩ s ≠ ∅ then
                 RS.add(s\res);
                 RS.replace(s, res ∩ s);
                 RS.replace(s,s\res);break;
        RS.add(res);
    Return the resource segment set RS;
```

B. Policy refinement

We build up a three-layer resource tree, in which the physical layer contains all the segments in a disjoint set, while the original policy effects on the logical layer. As shown in Fig. 3, RS1, RS2, RS3 and RS4 relate to (r_1), (r_1, r_3), (r_3) and (r_2), respectively. Based on this structure, we can assign rules to each resource segment in consistency with the original policy and refine policy on segment level.

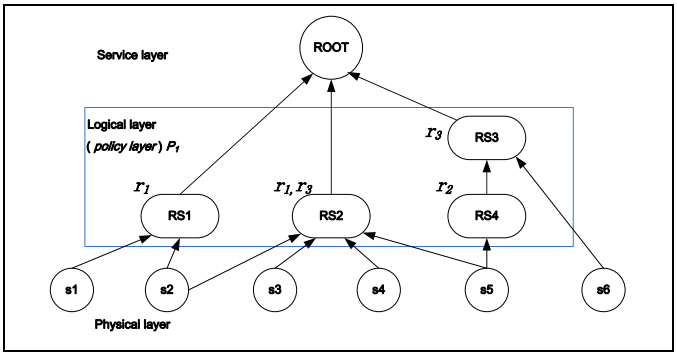


Figure 3. Resource tree.

We define the notation of *rule overlap* (denoting 'target matching' as '$\models$') and several RULEs used in the procedure of policy refinement.

Definition 2 (*Rule overlap*). On a single resource segment, let r_i , r_j be two parallel rules. If $\{\exists req : req \models r_i, req \models r_j\}$, then r_i overlaps with r_j (denoted by $r_i \propto r_j$), and the overlapped part is a rule pair Λ_{r_i,r_j}, consisting of Δ_{r_i} and Δ_{r_j}. Further, if $r_i \propto r_j$ and $r_i.effect = r_j.effect$, then $\Delta_{r_i} = \Delta_{r_j}$.

If deleting Δ_{r_i} (Δ_{r_j}) does not affect the final decision, then Δ_{r_i} (Δ_{r_j}) is removable in this policy. The following RULEs expound principles of removing redundant rules under different combining algorithms.

RULE 1 (CA = *Permit-Override*)

If $r_i \propto r_j$ and $r_i.effect = permit$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $r_i.effect = deny$, then Δ_{r_i} is removable.

RULE 2 (CA = *Deny-Override*)

If $r_i \propto r_j$ and $r_i.effect = deny$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $r_i.effect = permit$, then Δ_{r_i} is removable.

RULE 3 (CA = *First-Applicable*)

Assuming the sequence of r in policy is $seq(r)$.

If $r_i \propto r_j$ and $seq(r_i) < seq(r_j)$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $seq(r_i) > seq(r_j)$, then Δ_{r_i} is removable.

RULE 4 (CA = *Only-One-Applicable*)

If request on Λ_{r_i,r_j}, the decision is 'Not Applicable'.

If $r_i \propto r_j$, remove Δ_{r_i} and Δ_{r_j}.

The proofs of above RULEs are similar, and we takes RULE 1 for example, as shown in Fig. 4.

Proof

$\because r_i \propto r_j \therefore \exists \Lambda_{r_i,r_j} : \Delta_{r_i} \in r_i, \Delta_{r_j} \in r_j$.

If $r_i.effect = permit$, while the CA is Permit-Overrides, Δ_{r_j} will be shielded by Δ_{r_i}, no matter $r_j.effect$ is either 'permit' or 'deny'. Thus, removing Δ_{r_j} will not affect the decision, and Δ_{r_j} is removable.

If $r_i.effect = deny$, and if $r_j.effect = deny$, according to the Definition 2, we have $\Delta_{r_i} = \Delta_{r_j}$, remove any one of them is fine. However, if $r_j.effect = permit$, and CA is Permit-Overrides, so Δ_{r_i} will be shielded by Δ_{r_j}. Thus, in both conditions, removing Δ_{r_i} will not affect the decision, and Δ_{r_i} is removable.

Figure 4. Proof of RULE 1.

According to these RULEs, we propose the policy refinement algorithm, as illustrated in Algorithm 2. During the refining procedure, policy is projected to the physical layer, and rules might be refined, removed or kept still.

Algorithm 2: Policy refinement

```
// INPUT: a set of segments bound with rules, the policy CA.
// OUTPUT: refined set of resource segment.
Refine(G, CA)
  for each rule ∈ GetRule(policy) do //bind rules to segment
    for each s ∈ S do
      if s ⊂ GetRelatedResource(rule) then
        bind( rule, s);
  G ← S with bound rules on each element;
  for each g ∈ G do //refine rules on each segment
    for each pair(r_i, r_j) ∈ C^2_ruleSet do
      if r_i ∝ r_j then
      coupleSet.add( Λ_{r_i,r_j} ); //overlap of r_i, r_j
    for each Λ_{r_i,r_j} ∈ coupleSet do
      case CA= Permit-override then
        Execute by RULE 1;
      case CA = Deny-override then
        Execute by RULE 2;
      case CA = First-applicable then
        Execute by RULE 3;
      case CA = Only-one-applicable then
        Execute by RULE 4;
  Return the refined set G;
```

As for the situation of refining a policy set, the algorithm could be specified recursively for each children policy.

C. Algorithm performance analysis

We analyze the fine-grained policy optimization algorithm on computation overhead and storage overhead. Basically, we suppose that a policy *P* contains *K rules*, *N resource*s and *M resource segments*, which are generated by the data fragment algorithm, and on each segment *i*, there exist H_i *rules* and C_i *conflicts*.

1) Computation overhead

In the data fragment algorithm, it costs $O(N \log_2 M)$ to obtain resource segmentation set, where *M* varies upon the coupling degree among resources.

Nonetheless, in the procedure of policy refinement, the cost of policy projection is $O(KM)$, which is decided by the number of segments $\sum_{i \in M} s_i$ and related rules $\sum_{k \in K} r_k$. The overhead of refining an individual segment relies on finding rule conflict pairs, which contributes to $O(H_i \log_2 H_i)$, while resolving rule conflicts takes $O(L_i)$. Thus, the complexity of refining a resource segment set is the accumulation of cost on each element, resulting in $O(\sum_{i \in M} H_i \log_2 H_i + L_i)$.

Finally, the computation overhead of our approach is $O(N \log_2 M + KM + \sum_{i \in M} (H_i \log_2 H_i + L_i))$.

2) Storage overhead

We define the function $W(R)$ to measure the physical storage size. When we execute the algorithm, the overhead of storage is $\Theta\left(\sum_{i \in M} W(R_i) + \sum_{i \in M} H_i \cdot W(rule_i)\right)$.

Our approach advances in storage compared with original policies. In original policy, each resource is considered as a single entity, and the total size of all resources is $\sum_{i=1}^{N} W(R_i)$. However, by developing the relationship among resources, we extract the common parts of resources. As a result, the storage size is reduced by $\sum_{i=1}^{N} W(R_i) - \sum_{i=1}^{M} W(S_i)$.

IV. A CASE STUDY

The policy based access control methods can be applied in many fields such as banking, healthcare, ATM and market etc. to achieve data security and user privacy [17] [18] [19]. We apply the fine-grained policy optimization algorithm to data access control in cloud computing, and Fig. 5 describes the framework of a healthcare records management system.

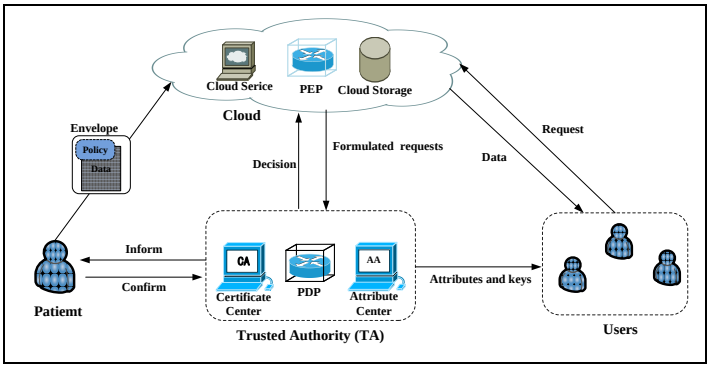


Figure 5. Application scenario on healthcare system.

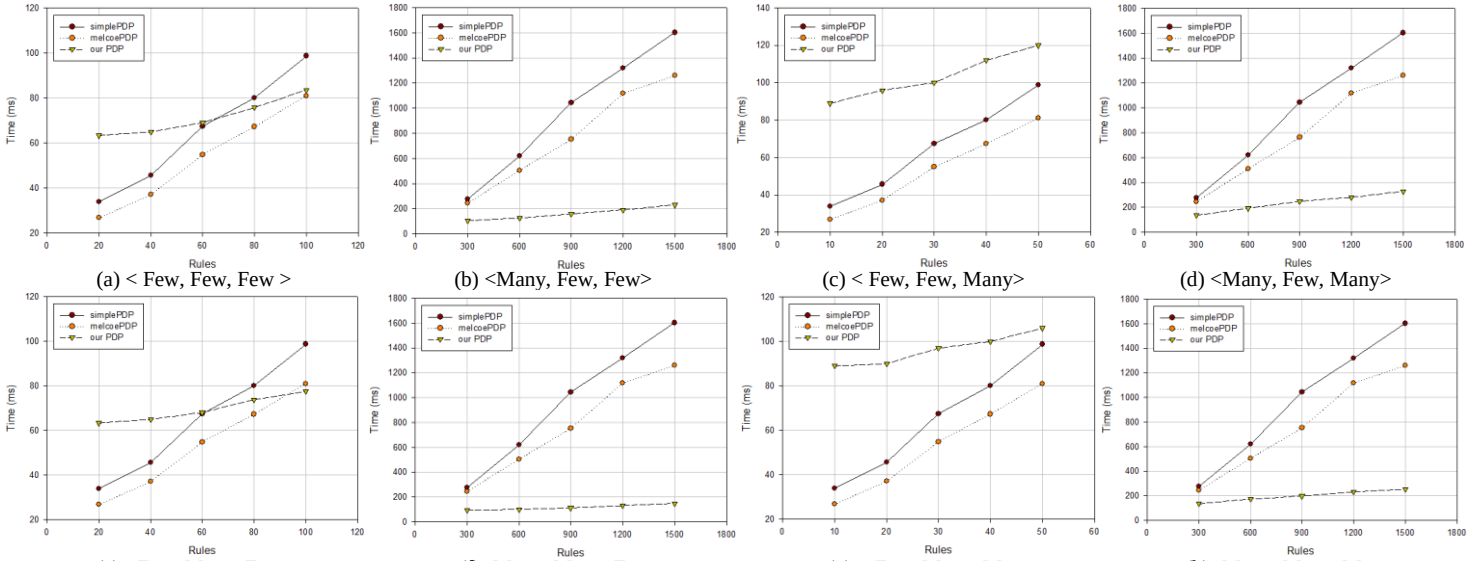
Assume that Alice is a patient, and she has well processed her private healthcare record with policies before outsourcing to the cloud. Bob is medical researcher in university, and he needs patient's health records for study. Once Bob requests to cloud for the statistics with his identity and public attributes, the cloud transforms the request into expressive XML format and sends it to the TA. Then, the PDP in TA makes the decision according to the defined policies and inform Alice of the result. Upon receiving Alice's acknowledgement, TA sends the decryption key to Bob through secure channels. Thus, Bob can have access to Alice private data under the conditions defined in the policies.

To evaluate the performance of our proposal, we utilize the policies with different amount of rules, different coupling degree of rules and resources. The coupling of rules leads to a certain number of conflicts, and the coupling of resources decides the number of resource segments. Our experiments were founded on the API of SUN-XACML and performed on Intel(R) Core(TM) i3-2330M CP U 2.30 GHz with 2.67-GB RAM running on Win7.

We generate synthetic policies for most situations and compare the decision efficiency with the existing methods, *Simple PDP* [3] and *Melcoe PDP* [20]. The *Simple PDP* adopts a list structure to traverse rules for matching, and *Melcoe PDP* employs category of data attributes. The statistics of two representative situations are listed in Table I and Table II. We use a triple <Few/Many, Few/Many, Few/Many> to simply express the amount of *rules*, *conflicts* and *segments*. Fig. 6 illustrates the experiment results of all the situations.

TABLE I. DECISION EFFICIENCY UNDER <FEW, FEW,FEW>

Policy parameters				**PDPs evaluation (*ms*)**		
Policy#	***Rule#***	***Res#***	***Seg#***	***Simple PDP***	***Melcoe PDP***	***Our PDP***
10	20	60	62	33.8608	26.7024	63.3365
20	40	60	62	51.6174	41.0145	65.0015
30	60	60	62	67.3090	54.8726	69.1087
40	80	60	62	82.0385	69.2740	75.7588
50	100	60	62	98.6908	81.0534	83.5972



(a) < Few, Few, Few > (b) <Many, Few, Few> (c) < Few, Few, Many> (d) <Many, Few, Many>

(e)< Few, Many, Few > (f)<Many, Many, Few> (g) < Few, Many, Many> (h)<Many, Many, Many>

Figure 6. Efficiency Impacts on amount of rules, conflicts and resource intersections.

TABLE II. DECISION EFFICIENCY UNDER <MANY, MANY,MANY >

Policy parameters				PDPs evaluation (*ms*)		
Policy#	***Rule#***	***Res#***	***Seg#***	***Simple PDP***	***Melcoe PDP***	***Our PDP***
10	300	60	155	276.5745	244.7562	135.6794
20	600	60	155	620.4870	509.0062	172.6407
30	900	60	155	1043.7546	761.0980	197.6577
40	1200	60	155	1319.8039	1117.6535	231.5092
50	1500	60	155	1602.8325	1259.0271	253.2671

Through the experiments, we conclude that the decision efficiencies of *Simple PDP* and *Melcoe PDP* depend on the amount of rules, with little concerning about the coupling degree of rules and resources. In the contrast, our approach has great advantages in the situation of large amount of rules. We also exceed traditional methods in multi-resource requests, since the redundancies of data are eliminated in the segmentation phase.

V. CONCLUSION

We have proposed an innovative mechanism of facilitating data security for cloud resource service. The fine-grained policy optimization algorithm projects the policy to the resource dimension, and refines rule on each individual resource segment. We can encrypt the sensitive data and attach sticky policy to ensure that the data is processed or handled according to customers' willing.

The fragmentation of resource decomposes data into obfuscated segments to protect the physical entities, while available services are provided in service layer and logical layer. Cloud users may request resources by names, without knowing the components or physical locations of the resources, and they can only get those identity-permitted data. We have discussed the performance of our proposal, in terms of the amount of rules, conflicts and segments. Through the experiment, we conclude that our approach has great advantages in large scale of policies.

We would develop a prototype and explore how our strategy can be applied to other fields concerning about access control and security.

ACKNOWLEDGMENT

This work was partially supported by the NSF of China under grants No. 61173048 and No. 61300041, Specialized Research Fund for the Doctoral Program of Higher Education under grant No. 20130074110015, and the Fundamental Research Funds for the Central Universities under Grant No.WH1314038.

REFERENCES

[1] W. Jansen and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing", NIST Special Publication, pp. 800-144, 2011.

[2] H. Takabi, J. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments", IEEE Security and Privacy, vol. 6, no. 6, pp. 24-31, 2010.

[3] S. Godik and T. Moses, "eXtensible Access Control Markup Language (XACML) Version1.1", OASIS, 2003.

[4] M. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services", Database and Expert Systems Applications (DESA), pp. 377-382, 2003.

[5] S. Pearson and M. Mont, "Sticky policies: an approach for managing privacy across multiple parties", IEEE Computer, vol. 44, no. 9, pp. 60-68, 2011.

[6] S. Trabelsi and J. Sendor, "Sticky policies for data control in the cloud", IEEE PST, pp. 75-80, 2012.

[7] G. Ahn, H.. Hu, J. Lee, and Y. Meng, "Representing and reasoning about web access control policies", IEEE Software and Applications, pp. 137-146, 2010.

[8] H. Hu and G. Ahn, "Discovery and resolution of anomalies in web access control policies", IEEE Dependable and Secure Computing, vol. 10, no. 6, pp. 341-354, 2013.

[9] Y. Wang, D. Feng, and L. Zhang, "XACML policy evaluation engine based on multi-level optimization technology", Journal of Software, vol. 22, no. 2, pp. 323−338, 2011.

[10] M. Said, M. Shehab, and S. Anna, "Adaptive reordering and clustering-based framework for efficient XACML policy evaluation", IEEE Service Computing, vol. 4, no. 4, pp. 300-313, 2011.

[11] D. Lin, P. Rao, R. Ferrini, and E. Bertino, "A similarity measure for comparing XACML policies", IEEE Knowledge and Data Engineering, vol. 25, no. 9, pp. 1946-1959, 2013.

[12] A. Bertolino, S. Daoudagh, and F. lonetti, "Automated testing of eXtensible Access Control Markup Language-based access control systems", IET Software, vol. 7, no. 4, pp. 203-212, 2013.

[13] D. Agrawal, J. Giles, K. W. Lee, and J. Lobo, "Policy ratification", IEEE Policies for Distributed Systems and Networks, pp. 223-232, 2005.

[14] X. Wu and P. Qian, "A verification for PDAC model by policy language", ICCSE, pp. 14-17, 2012.

[15] G. Bruns, D. Dantas, and M. Huth, "A simple and expressive semanticframework for policy composition in access control", Formal methods in Security Engineering, ACM, pp. 12-21, 2007.

[16] C. Ngo, Y. Demchenko, and C. D. Laat, "Decision Diagrams for XACML PolicyEvaluation and Management", Computers & Security, vol. 49, no. 1, pp. 1-16, 2015.

[17] M. Ghorbel, A. Aghasaryan, and M. P. Dupont, "A multi-environment application of privacy data envelopes", Policies for Distributed Systems and Networks, pp. 180-181, 2011.

[18] M. Li, S. Yu, and Y. Chen, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption", IEEE Parallel and Distributed Systems, vol. 24, no. 1, 131-143, 2013.

[19] Asela, "Banking sample with XACML", http://xacmlinfo.org/2014/03/11/atm-banking-sample-with-xacml/, 2014.

[20] Jajodia and P. Samarat, "A logical language for expressing authorizations", IEEE Security and Privacy, pp. 31-42, 1997.